In S. Koenig and R. Holte (Eds.), Abstraction, Reformulation, and Approximation (pp. 99-115). 2002: Springer-Verlag.

# On the Construction of Human-Automation Interfaces by Formal Abstraction

Michael Heymann[1], and Asaf Degani [2]

[1] Department of Computer ScienceTechnion, Israel Institute of Technology
heymann@cs.technion.ac.il

[2] NASA Ames Research Center, California
adegani@mail.arc.nasa.gov

**Abstract**. In this paper we address the problem of designing systems for human-automation interaction that insure satisfaction of a wide range of performance requirements (such as guaranteeing the safety and liveness of mission critical operations). Our approach is based on formal procedures that focus on the information provided to the user. We propose a formal methodology for constructing interfaces and corresponding user- manuals that is based on performing a systematic abstraction of the behavioral model of the system. The procedure is aimed at achieving two objectives: First, the interface must be correct in that with the given interface the user will be able to perform the specified tasks correctly. Secondly, the interface must be succinct. The paper discusses the underlying concepts and the formal methods for this approach. Two examples are used to illustrate the methodology. The algorithm for constructing interfaces that is proposed in the paper can be automated, and a preliminary software system for its implementation has been developed.

## 1 Introduction

Human interaction with automation is so widespread that almost every aspect of our lives involves computer systems, information systems, machines, and devices. These machines are complex and are comprised of many states, events, parameters and protocols. User interfaces for such machines always present a (highly) reduced description of the underlying machine's behavior.

In the majority of today's automated systems, the human is the supervisor. Users interact with systems or tools to achieve specified operational tasks (Parsuramann et al., 2000) such as the execution of specific sequences of actions (e.g., a procedure for setting up a medical radiation machine), monitoring a machine's mode changes (e.g., an automatic landing of an aircraft), or preventing a machine from reaching specified illegal states (e.g., tripping a power grid). To achieve these task specifications, the user is provided with information about the behavior of the machine by means of an interface and associated user-manuals and other training material.

Naturally, for the user to be able to interact with the machine correctly and reliably so as to achieve the task specification, the information provided to the user about the machine must first and foremost be correct. Yet, while correct interaction can, in principle, always be achieved by providing the user with the full detail of the machine behavior, the amount of detail is generally unmanageable. Therefore, in practice, the interface and related user manuals are always a reduced, or abstracted, description of the machine's behavior, and a major concern of designers of automated systems is to make sure that these abstracted interfaces and manuals are adequate and correct.

Currently, the design decisions as to what information must be provided to the user, both in the interface and in user-manuals, are made intuitively. Systematic methodologies do not exist for these decisions and the resultant interfaces are sometimes either overly complex or flawed, leading to what is commonly called "automation surprises," where operators (e.g., pilots, technicians, users) have difficulty understanding the current status of an automatic system as well as the consequences of their interaction with it (Woods, Sarter, and Billings, 1997).

In an earlier paper (Degani and Heymann, 2002), we discussed a methodology for evaluating interfaces and user manuals. Given a description of the machine, specifications of the user's task, interface, and all relevant information the user has about the machine, the procedure evaluates whether the interface and user manual information are correct for the task. The proposed procedure can be automated and applied to the verification of large and complex human-machine systems.

In the present paper we take an additional step and discuss a formal methodology for automatic generation of correct and succinct interfaces and user manuals.

## 2 Formal Aspects of Human-automation Interaction

We focus primarily on the information content provided to the user about the behavior of a system. This aspect of user interaction with machines can be described and analyzed formally by considering the following four elements: (1) the machine-model, (2) the operational tasks, (3) the machine's interface with the user, and (4) the user's model of the machine, i.e., the information provided to the user about the machine behavior (e.g., in the user manual).

### 2.1 Machine

The machines are modeled as finite state transition systems. A *state* represents a mode, or configuration, of the machine. Transitions represent discrete-state (mode) changes that occur in response to events that trigger them. Some of the transitions occur only if the user triggers them, while other transitions occur automatically and are triggered by the machine's *internal* dynamics, or its *external* environment.

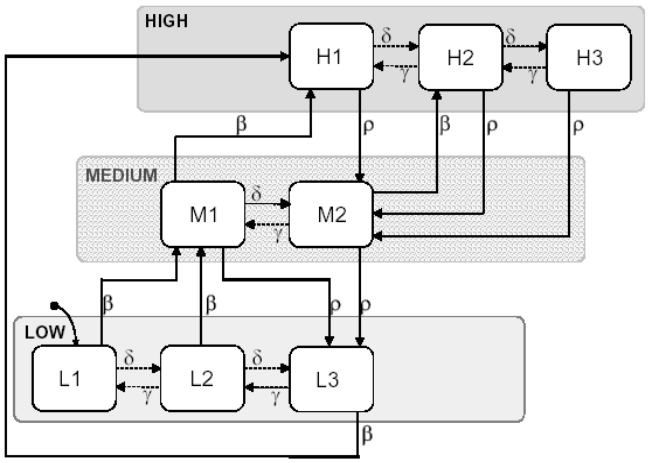


**Figure. 1.** Transmission system.

To illustrate a typical machine model, let us consider the machine of Figure 1, which describes a simplified multi-mode three-speed transmission system proposed for a certain vehicle. We use the convention that user-triggered transitions are described by solid arrows, while automatic transitions are depicted by dashed arrows. The transitions are labeled by symbols to indicate the (triggering) circumstances under which the machine moves from state to state. The transmission has eight states, or modes. These modes are grouped into three super-modes that represent manually switchable gears (or speeds): low, medium and high. The states within each speed represent internal torque-level modes. Thus there are torque modes $L1, L2, L3$, in the low speed super mode; there are torque modes $M1, M2,$ in the medium speed super mode; and modes $H1, H2, H3,$ in the high speed super mode. The transmission shifts automatically between torque modes (based on torque, throttle, and engine and road speeds). The automatic up-shifts (to higher torque modes) are denoted by the event symbol $\delta$ and the automatic down-shifts by the symbol $\gamma$. The (user operated) manual speed changes, achieved by pushing a lever up or down, are denoted in the Figure by the event symbols $\beta$ and $\rho$, respectively. Pushing the lever up shifts to a higher speed and pushing down shifts to a lower speed. The transmission is initialized in the low torque mode $L1$ of the low speed (as indicated in the Figure by the free incoming arrow).

### 2.2 Task Specifications

The second element is the specification of the operational tasks the user is required to perform while using the machine. For example, a common task specification in an

automated control system is that the user be able to determine unambiguously the current and the subsequent mode of the machine.

In terms of a formal description, the task specification to which we confine our attention in the present paper consists of a partition of the machine's state-set into disjoint clusters that we shall call specification classes (or modes) that the user is required to track unambiguously. In other words, does the user know whether the system is currently in, or is about to enter into, the super-mode High, Medium, or Low? We note that the user is not required to track every internal state change of the machine: for example, transitions between the modes $L1$, $L2$ and $L3$ inside mode Low.

### 2.3 Interface

The third element is the user interface. In practice, the interface consists of a control unit through which the user enters commands (e.g., mode selections, parameter changes) into the machine, as well as a display through which the machine presents information to the user. Generally, the interface provides the user a simplified view of the machine. Not all the events of the machine are annunciated to the user, and the interface displays only partial information about the actual behavior of the machine.

Formally, the interface consists of a listing and description of the events accessible to the user. These include, of course, all the user-triggered events (inputs to the machine), but generally only a subset of the events that are associated with automatic transitions. This is because some of the latter are not monitored at all, and others are monitored only in groups. The interface annunciation tells the user only that one of the events in the group took place, without specifying which.

To illustrate, let us return to the multi-mode transmission model of Figure 1. The system in Figure 2 gives one possible user interface for this model. Here the monitored events are only the ones triggered by the user. In the Figure 2 we have also provided a description of the three display modes, as well as how the user would observe the machine's behavior when all automatic transitions are internalized and unobserved. Note that the torque modes are completely suppressed from view.

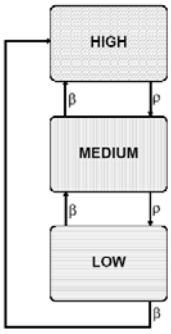


**Figure 2.** Proposed interface and user model.

### 2.4 User model.

As mentioned earlier, the interface provides the user with a simplified view of the machine, in that it displays only partially the machines internal behavior. The description of the machine's operation that is provided to the user is generally also an abstracted simplification of the actual machine behavior. This description is usually provided in terms of a user manual, training material, formal instruction, or any other means of teaching the user; however, it is presented here as a formal model that we refer to as the *user model* of the machine. By its very nature, the user-model is based on the interface through which the user interacts with the machine, and thus relates to the modes and events that are displayed there. Therefore, for analysis purposes the interface events and modes are all explicitly referred to in the user-model, and in this respect can be thought of as "embedded" in the user-model.

Let us examine the user interface displayed in Figure 2. This Figure depicts a possible user-model associated with the interface that monitors only the user-triggered events of the transmission system. This particular user-model has been obtained from the machine model of Figure 1 by suppressing (internalizing) the events that are not monitored, and grouping the states as suggested by the specification. It can be seen that the manual shifts from *MEDIUM* up to *HIGH* or down to *LOW*, as well as the down-shift from *HIGH* to *MEDIUM*, are always completely predictable. However, the up-shift from the LOW gear depends on the current torque mode. Note that the up-shifts from L1 and L2 switch the transmission to *MEDIUM* speed, while the up-shift from L3 switches the transmission to the *HIGH* speed. Therefore, from the suggested interface of Figure 2, it cannot be predicted whether the up-shift will lead the transmission from *LOW* to *MEDIUM*, or to *HIGH* gear. We must conclude that the user-model is inadequate for the task.

An alternate user-model for the transmission model that may remedy the above mentioned problem is presented in Figure 3. This user-model describes an interface that also monitors the occurrences of two specific automatic transitions, in addition to all user-actuated events. This user-model, in particular, is aimed at enabling the operator to determine whether the transmission is in a display-mode *LOW-1* (where an up-shift is supposed to lead to *MEDIUM* speed), or in the display-mode *LOW-2* (where an up-shift leads to *HIGH*).

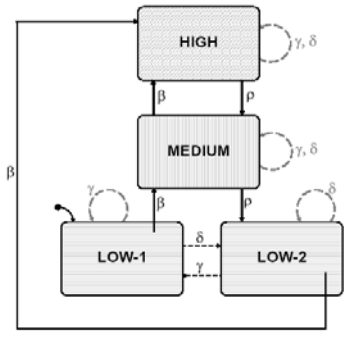


**Figure 3.** Alternate interface and user model.

However, although the alternative user model of Figure 3 appears to have solved the problem, a formal verification employing the methodology recently proposed by Degani and Heymann (2000; 2002) shows that this user-model is also inadequate.

It is of course possible to try out other interfaces and user-models and then employ the verification procedure to determine their correctness. However, such an approach is not likely to be very fruitful: It may take considerable effort to develop and verify one design after the other, with no guarantee of success. Furthermore, even when a correct interface is found, there is no assurance that it is the simplest.

## 3 Machine Model Reduction

As mentioned earlier, one possible choice of user model is to take the full machine model as user model and the complete machine event set as the set of monitored events. If the machine model is deterministic (as we assume throughout this paper), this will insure that there will never be any problem in predicting the next state of the machine. But the operator would be required to track every state and every event in the machine – a formidable and impractical job. In the simple example of Figure 1, the machine has 8 states, 18 transitions and 4 distinct transition labels. But this is a tiny number when compared to "industrial size" situations.

In the present section we shall describe a procedure for the generation of *all* optimal user models and interfaces for a given machine model and task specification. In particular, we shall consider the problem of constructing, for a given machine and task specification, the set of all *best* possible user-models and event abstractions that satisfy the specification. Here, by *best* user models and interfaces we mean the ones that cannot be further reduced! Since, as we shall see, these user models (and associated event abstractions) are generally not unique, we cannot speak of user-model "synthesis," but rather, of machine model *reduction.* We shall show how all "smallest" user models and associated interfaces can be derived.

### 3.1 Compatible state sets and covers

We assume that the machine-model is given as a state machine and that the task specification is given as a partition of the state-set into disjoint classes of states that we refer to as *specification classes* (Heymann and Degani, 2002). Thus, each state of the machine model belongs to a unique specification class. (In Figure 1 which depicts the multi-mode three speed transmission, the specification classes consist of the three speeds; Low, Medium and High. Each state, or mode, belongs to exactly one speed.)

Let us consider a machine-model given as a state-machine, and let the task specification consist of a partition of the machine-model's state set $Q$ into disjoint specification classes $Q_1,...,Q_l$ (as described, for example, in Figure 1 where $l = 3$).

The user model must enable the user to operate the system correctly with respect to the specification classes. That is, it must enable the user to track the specification classes but not necessarily individual states. Thus, the user does not need to be able to

distinguish (by means of the user model and interface) between two states $p$ and $q$ of the same specification class, if for the purpose of tracking the specification classes unambiguously it is sufficient for the user to know that the machine visited *either* $p$ or $q$. More explicitly, the user does not need to be able to distinguish between $p$ and $q$ if the specification class visited following any user-machine interaction starting in state $p$, is the same as the specification class visited following the same user-machine interaction starting at state $q$. This leads to the following definition: Two states, $p$ and $q$, are *specification equivalent* (or *compatible*), if given that the machine is presently in either state $p$ or $q$ (of the same specification class), the specification classes to be visited under future inputs will be the same. Stated more formally, we have

**Definition:** Two states $p$ and $q$ are specification compatible if and only if the following two conditions both hold:

1. The states $p$ and $q$ belong to the same specification class,
2. If $p'$ and $q'$ are states such that there exists an even-string $s = \sigma_1 ... \sigma_n$ for which $p \xrightarrow{s} p'$ and $q \xrightarrow{s} q'$ are both defined, then $p'$ and $q'$ belong to the same specification class.

It is clear that if the only concern is to track the specification classes, two specification compatible states need not be distinguished in the user model. We may also conclude immediately that any set of states is specification compatible if all the pairs of states within that set are specification compatible.

Thus, if an efficient procedure is found for computation of all specification compatible pairs, the set of all compatible state sets will easily computed. Indeed, the compatible triples will be obtained as the state triples, all of whose pairs are compatible; compatible quadruples as the quadruples all of whose triples are compatible, and so on.

Next, we have the following:

**Definition:** A set C of compatible sets of states is called a cover of the state set of the machine-model, if every state of the machine-model is contained in one or more elements of C.

Since a set that consists of a single state is (trivially) compatible, it follows that every state is included in at least one compatible set, so that the set of all compatibles is always a cover.

**Definition:** A compatible set of states is called a *maximal* compatible set, if it is not a proper subset of another compatible set; that is, if it is not contained in a bigger compatible set of states.

Since sets that consist of a single state are compatible, it is clear that every state is contained in at least one maximal compatible set. It follows that the set of maximal compatibles is a cover.

**Definition:** A cover $C$ of compatibles is called a *minimal* cover, if no proper subset of $C$ is a cover.

Of particular interest to us will be the set of all minimal covers formed from the set of maximal compatibles. That is, we shall be interested in minimal covers whose component elements are maximal compatible sets. In general, the number of such minimal covers can be greater than one.

We shall see below that minimal covers by maximal compatibles constitute the foundation of the model reduction and interface generation procedure. However, we shall first show the set of compatibles is computed.

### 3.2 Generation of compatible pairs

As stated above, the computation of compatible sets hinges on the construction of the set of all compatible pairs. An efficient iterative algorithm for construction of compatible state pairs is based on the use of merger tables (see e.g., Paull and Ungar 1959, and Kohavi 1978, where related model reduction problems are discussed).

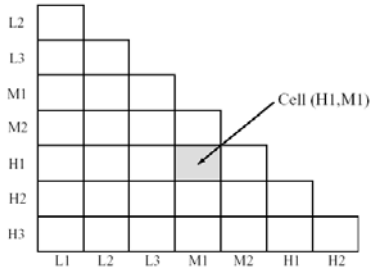


**Figure 4.** Table of all pairs

A merger table is a table of cells representing distinct state pairs. An initial table for the eight states of our transmission example is shown in Figure 4. Each cell of the table corresponds to a pair of distinct states, and each pair of distinct states appears in the table exactly once.

Next, we have the following observations that can be easily derived from the definition of compatible pairs:

A state pair $(p,q)$ of the same specification class is *compatible* if and only if for every event symbol $\sigma$ such that $p \xrightarrow{\sigma} p'$ and $q \xrightarrow{\sigma} q'$ are both defined, it is true that either $p' = q'$, or the pair $(p',q')$ is compatible.

We shall use the above characterization of compatible sets to obtain a complementary characterization of all pairs that are **not** compatible (or incompatible). It will then be convenient for us to compute recursively the set of all incompatible pairs. The set of compatible pairs will then consist of all state pairs that are not found to be incompatible. Based on the above characterization of compatible pairs, the characterization of incompatible pairs is as follows:

A state pair $(p,q)$ is *incompatible* if and only if either $p$ and $q$ belong to distinct specification classes, or there exists an event symbol $\sigma$ for which $p \xrightarrow{\sigma} p'$ and $q \xrightarrow{\sigma} q'$ are both defined, and the state pair $(p',q')$ is incompatible.

Using the above observations regarding compatible and incompatible pairs, the determination as to whether a state pair is compatible or incompatible is computed iteratively as follows.

1. For each state pair $(p,q)$ that can be determined as incompatible in the first step based on the above characterization (i.e., if $p$ and $q$ belong to distinct specification classes), we mark the corresponding cell *F* (for false). For all other state pairs, we write in their cells their associated *transition pairs* that consist of all distinct state pairs $(p',q')$ for which there exists an event symbol $\sigma$, such that the transitions $p \xrightarrow{\sigma} p'$ and $q \xrightarrow{\sigma} q'$ are both defined.

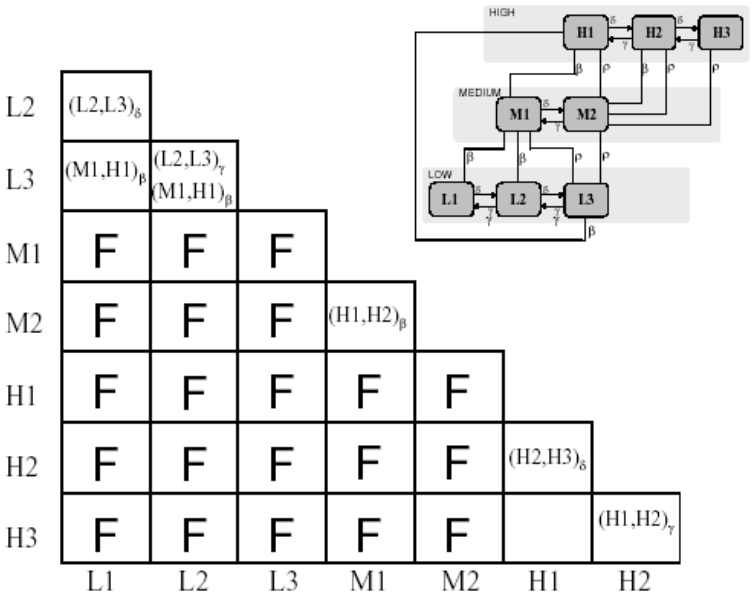


**Figure 5.** Resolution table (initial).

For illustration, the initial resolution table for the transmission model of Figure 1 is presented in Figure 5. Notice that each transition pair in the table has been subscripted with the associated event label. This subscription is not essential to the algorithm and is for the reader's convenience only. Notice further that the cell *(H1,H3)* is empty because it is neither incompatible nor has associated transition pairs. Next, the table is resolved iteratively.

2. At each step of the iteration every state pair that has not yet been determined as *F* is updated as follows: If the cell of a state pair $(p,q)$ includes a transition pair $(p',q')$ whose cell has already been determined as *F* (incompatible), then the cell of $(p,q)$ is also denoted *F*. Otherwise, the cell of $(p,q)$ is modified as follows: Each transition pair $(p',q')$ in the cell of $(p,q)$ is replaced by all the transition pairs that appear in the cell of $(p',q')$.
3. If in a given iteration step no new incompatible state pairs are found (i.e., no new *F* designations are added to the table), then all the state pairs that are not designated as *F*, are given the designation *T* (for true). This completes the table resolution procedure and the determination of all compatible pairs.

To illustrate the iteration steps of the procedure, let us return to our transmission example. The table of Figure 6 is obtained from that of Figure 5 as follows: First we replace the transition pairs in the cell *(L1,L2)* by those in the cell *(L2,L3)*. The cells *(L1,L3)* and *(L2,L3)* are denoted with *F* because their cells include incompatible pairs. The remaining undecided state pairs (those that have not yet been given the value *F*) are modified according to the algorithmic procedure. For example, in the cell *(M1,M2)* we list the transition pairs from the table of Figure 5 of the cell *(H1,H2)* that consists of *(H2,H3)*.

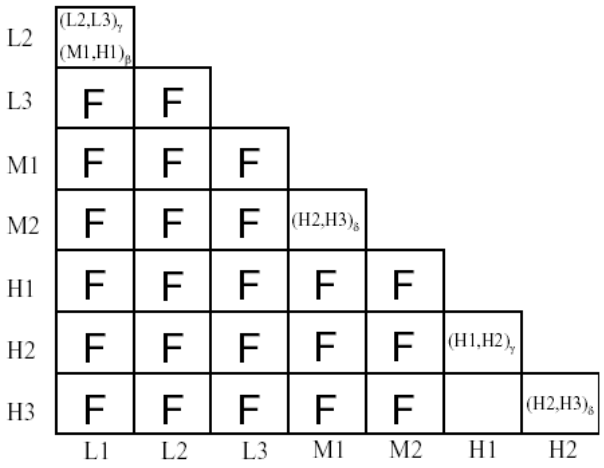


**Figure 6.** Resolution table (after first iteration).

In the next resolution step the table of Figure 7 is obtained. Here the cell (L1,L2) is marked F upon substituting the value F of the cell (M1,H1,) which is incompatible. The remaining undecided cells are modified as specified by the algorithm. In fact, notice that no further change needs to be made to the table.

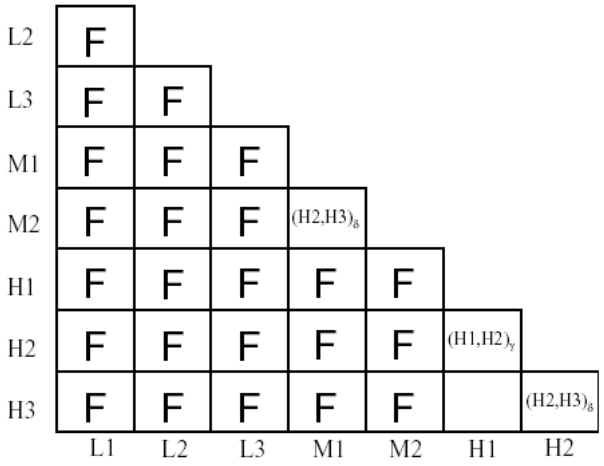


**Figure 7.** Resolution table (after second iteration).

In the next step, no further incompatible pairs are created and the table remains identical to that of Figure 7. At this point, all the remaining undecided cells are marked T a shown in the table of Figure 8, concluding the table resolution.

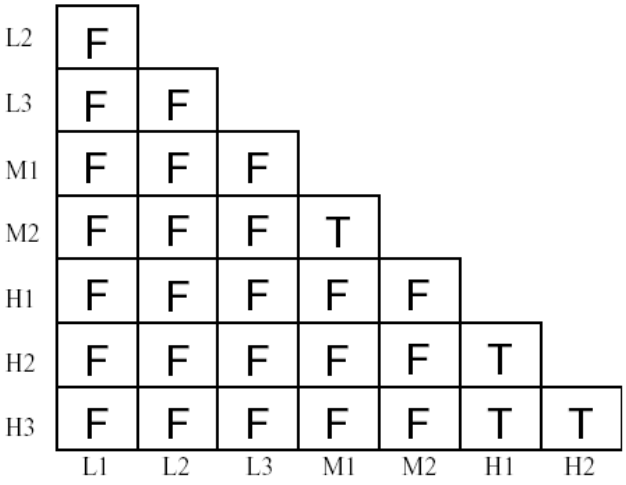


**Figure 8.** Resolution table (completed).

Thus, as seen in Figure 8, for the example of Figure 1, the set of compatible pairs consists of *(M1,M2), (H1,H2), (H1,H3), and (H2,H3).* Notice that the states *L1, L2* and *L3* do not appear in any compatible pairs and therefore the singleton sets *(L1), (L2)* and *(L3)* are clearly maximal compatibles.

### 3.3 Generation of the set of maximal compatibles

The procedure for generation of maximal compatibles consists of first systematically creating all compatible sets. We begin by computing all compatible triples, then compatible quadruples, then quintuples, and so on. A compatible triple is a triple all three of whose pairs are compatible; a compatible quadruple is a quadruple all of whose pairs are compatible, which is equivalent to a quadruple whose four triples are all compatible, and so on. Once all compatibles are listed, the maximal ones can easily be computed by deleting from the list all compatibles that are contained within larger ones.

For the transmission example, the maximal compatibles are easily found to be the sets *(L1), (L2), (L3), (M1,M2)* and *(H1,H2,H3).* It is also not difficult to see that, in this case, they partition the state set into disjoint subsets and hence form the (unique) minimal cover by maximal compatibles.

### 3.4 Generation of reduced models

The generation of a reduced model that can serve as a correct user model for the given machine and specification is based on an abstraction of the machine-model. This reduced model is obtained by clustering the states into sets that consist of a minimal cover by maximal compatibles.

To this end, let us assume that a minimum cover consists of a given set of maximal compatibles $C_1,...,C_l$, where the set $C_i$, $i=1,...,l$, consists of states $\{q_{i_1},...,q_{i_{n_i}}\}$ of the machine model. The maximal compatibles $C_1,...,C_l$ form the state set of the reduced model. Here it is noteworthy that a minimal cover by maximal compatibles need not be a partition of the state set into disjoint subsets. Specifically, while each state of the machine model must be contained within some maximal compatible set, it may well be the case that a state is contained in more than one maximal compatible of the minimal cover. That is, these sets may (sometimes) have overlaps.

Next, we turn to computing the transitions in the reduced model. An event symbol $\sigma$ is said to be active at $C_i$, if there exists an outgoing transition in the machine model labeled by $\sigma$, at some state $q \in C_i$. That is, there exists a state $q'$ in the machine model, such that $q \xrightarrow{\sigma} q'$ is defined. We denote by $C_i(\sigma)$ the set of all states $q \in C_i$ for which an outgoing transition labeled by $\sigma$ exists.

Next, we define $S_i(\sigma)$ to be the set of all states $q'$ of the machine model, such that $q \xrightarrow{\sigma} q'$ for some $q \in C_i(\sigma)$. Thus, the set $S_i(\sigma)$ is the set of all states of the machine model that can be reached from states in $C_i$ through the event $\sigma$. It readily follows from the definition of compatible sets that there exists one or more element of $C_1,...,C_l$ which contain $S_i(\sigma)$. In the reduced model we then create a transition labeled by $\sigma$ going from the state $C_i$ to the state $C_j$, where $C_j$ is the maximal compatible that contains $S_i(\sigma)$. If more than one such set $C_j$ exists, we can choose any one of these (and to avoid non-determinism in the reduced model we choose exactly one).

To summarize, the reduced model associated with the minimal cover $C_1,...,C_l$ is obtained as follows. The state set of the reduced model consists of elements $p_1,..., p_l$ (think of $p_i$ as associated with $C_i$). There is a transition labeled $\sigma$ from $p_i$ to $p_j$ if $C_j$ is the (chosen) set that contains $S_j(\sigma)$. The reduced model is initialized at state $p_k$ if the machine model is initialized at a state in $C_k$ (where, as before, there may be more than one possible selection if the initialization state is contained in more that one of the $C_i$). The reduced model obtained for the transmission example is shown in Figure 9.

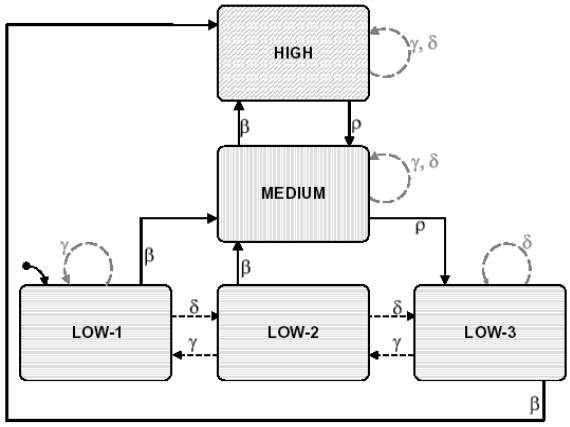


**Figure 9.** The reduced user model.

### 3.5 Event Abstraction

The final step of the model reduction procedure consists of the abstraction of the reduced model's event set (when possible). Specifically, we ask which events can be internalized (i.e., need not be monitored) and which events can be clustered into

groups so that instead of being monitored individually, they be monitored collectively. That is, the user will be informed that some events in the group occurred, but will not be informed which events of the group actually took place.

To this end the following abstraction rules apply:

1. An event can be internalized if it occurs in the reduced model only in self-loops.
2. A set of events can be grouped together, if every state transition that can be triggered by any event of the group can also be triggered by any other event of the group.

In the transmission example no event abstractions are possible. An illustration of event abstractions is provided in the example of the next section.

## 4 An Abstract Machine Example

In the above discussion on machine model reduction, we used an example of a transmission system. In this final section, we shall apply the reduction algorithm to a somewhat more complex machine. The machine in Figure 10 has nine states and 25 transitions. There are three specification classes: the gray region that includes states 7, 8, and 9; the wave-like region that harbors state 4 and 6; and the rest of the states of the machine (1, 2, 3, and 5). The task specification is similar to our previous one: the user has to track the machine along these three regions (or modes). Specifically, the user must be able to identify the current mode of the machine and anticipate the next mode of the machine as a consequence of his or her interactions.

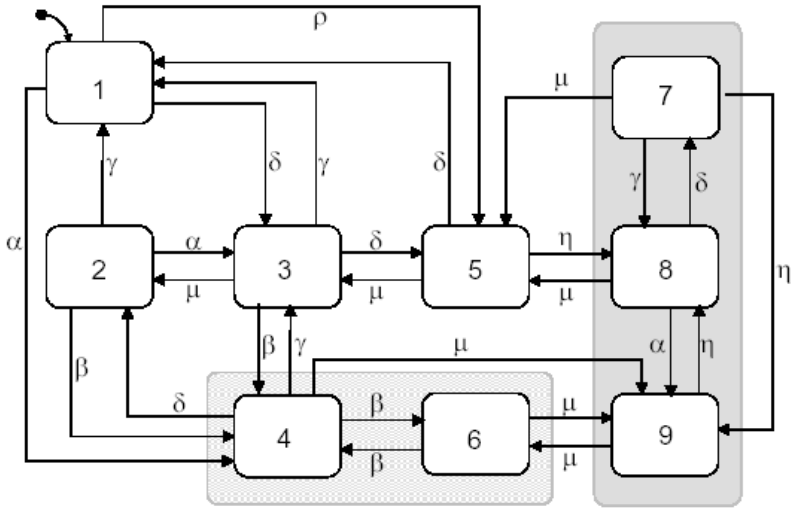


**Figure 10.** An abstract machine model.

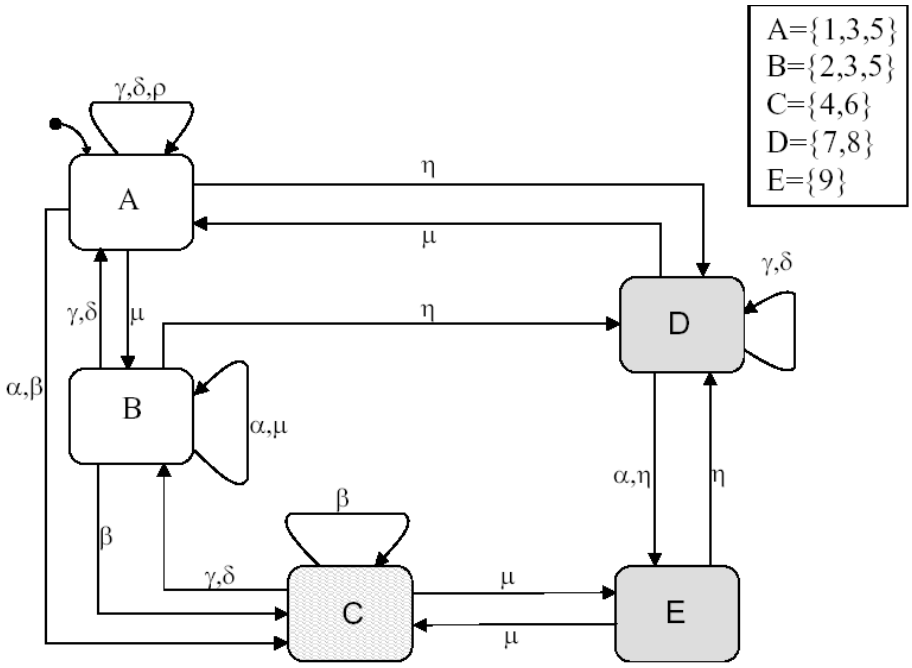


**Figure 11**. Reduced model.

We perform the reduction procedure along the steps described in the previous section. First the table is constructed, and then the iterations are performed. The procedure terminates with only one minimal cover of maximal compatibles that consists of four state sets: (1,3,5) (2,3,5) (4,6) (7,8) and (9). Notice however, that this example illustrates a case in which the cover is not a partition of the state set. Indeed, the state 3 is included in two distinct maximal compatibles.

We then arbitrarily assign names to these sets, and call them A, B, C, D, and E, respectively. The reduced machine is obtained upon computation of the abstracted transitions as explained earlier (see Figure 11). It can be seen in this figure that the event $\rho$ occurs only in the self-loop in state A, and that the events $\gamma$ and $\delta$ are interchangeable. Thus, $\rho$ can be internalized and the events $\gamma$ and $\delta$ can be grouped. The result of this event abstraction is presented in the final reduced (user) model of Figure 12, which contains only 5 states and 16 transitions.

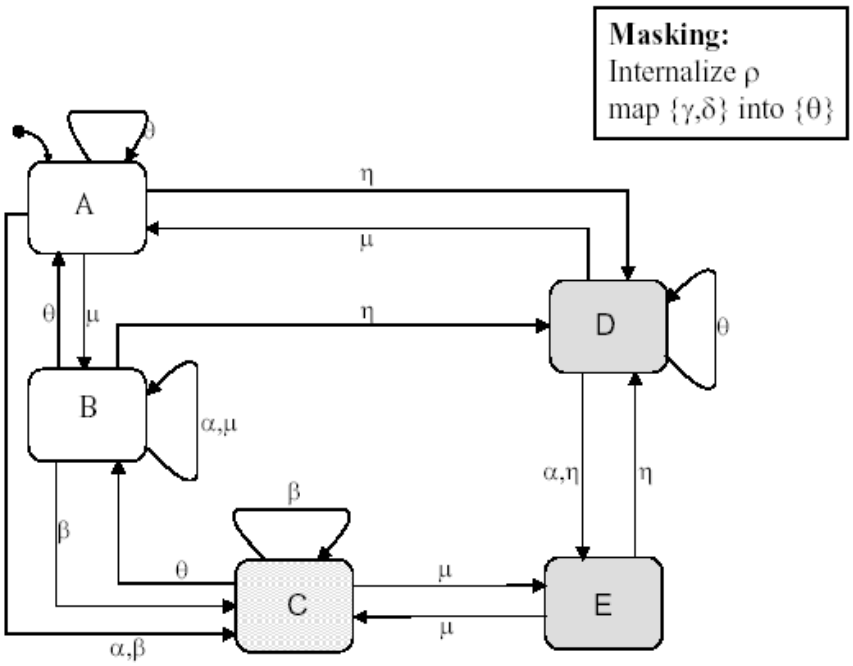


Figure 12. Reduced model (with masking and internalization of events.

## 5 Conclusions

In this paper we discussed several formal aspects of the design of human-automation interaction. Specifically, we focused attention on the construction of user models and interfaces. Two objectives guided us in our design and analysis: (1) that the interfaces and user models be correct; and (2), that they be as simple as possible. We described a systematic procedure for generating such correct and succinct user-models and interfaces.

The proposed reduction procedure generates interfaces that are not necessarily intuitive or easily correlated with the underlying system (e.g., see the reduced user model of Figure 12). Nevertheless, these user models are correct and efficient. They are also, irreducible.

The proposed procedure may lead to more than one possible minimal (irreducible) interface and user-model. That is, it may find several minimal covers (of maximal compatibles). These minimal covers are all correct and efficient reductions of the same machine and task-specification. Naturally, the decision as to which one is selected constitutes a human-factors and/or engineering design decision. It affords the designer with several candidate interfaces and allows designers the freedom to choose the most appropriate one, given other design considerations such as Graphical User Interface considerations, users' preferences, and ease of implementation.

## References


Degani, A. and Heymann, M., Meyer, G., and Shafto, M. (2000). *Some Formal Aspects of Human-Automation Interaction, NASA Technical Memorandum 209600*, NASA Ames Research Center, Moffett Field, CA.

Degani, A. and Heymann, M. (2002). Formal Verification Of Human-Automation Interaction. *Human Factors*.

Heymann M., and Degani A. (2002). *On abstractions and simplifications in the design of human-automation interfaces.* NASA Technical Memorandum, NASA Ames Research Center, Moffett Field, CA.

Kohavi, Z. (1978). *Switching and Finite Automata Theory*. New York: McGraw-Hill.

Parasuraman, R., Sheridan, T.B., and Wickens, C.D. (2000). A model for the types and levels of human interaction with automation. *IEEE Transaction on Systems, Man, and Cybernetics – Part A: Systems and Humans, 30*(3), 286-297.

Paull, M.C. and Unger, S.H. (1959). Minimizing the number of states in incompletely specified sequential switching functions. *Institute of Radio Engineers Transactions on Electronic Computers*, 356-367.

Woods, D., Sarter, N., and Billings, C. (1997). Automation surprises. In G. Salvendy (Ed.), *Handbook of human factors and ergonomics* (pp. 1926-1943). New York: John Wiley.